REMARKS

This Amendment is intended to fully respond to the Final Office Action mailed February 2, 2006. In that Office Action, claims 1, 2, 4, 6-22, 24, 26, 27, and 29-33 were examined, and all were rejected. More specifically, claims 1, 2, 6, 9, 11-20, 24, 26, 27, 29, and 31 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Burridge (USPN 6,918,106), hereinafter "Burridge," in view of "Enterprise JavaBeans," published by Haefel, hereinafter "Haefel"; claims 4 and 10 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Burridge in view of Haefel as applied to claim 1, and further in view of "Method to Update Java Class Library in Client Computer at Runtime," published by IBM, hereinafter "IBM"; claims 7, 8, 30, 32, and 33 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Burridge in view of Haefel as applied to claims 1 and 26, and further in view of Chan (USPN 6,470,494), hereinafter "Chan"; and claims 21 and 22 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Burridge in view of Haefel as applied to claim 1, and further in view of Menachemi (U.S. Patent Application Publication 2002/0103810), hereinafter "Menachemi." Reconsideration of these rejections, as they might apply to the original and amended claims in view of these remarks, is respectfully requested.

In this Amendment, claims 1, 9, 24, and 26 have been amended; no claims have been canceled; and new claims 50 and 51 have been added. Therefore, claims 1, 2, 4, 6-22, 24, 26, 27, 29-33, 50, and 51 are present for examination.

Claim Rejections – 35 U.S.C. § 103

Claims 1, 2, 6, 9, 11-20, 24, 26, 27, 29, and 31 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Burridge in view of Haefel.

Applicants respectfully traverse the section 103 rejections for all claims. The claims as amended preclude a finding of a prima facie case of obviousness because one or more of the requirements of a prima facie case is absent. Indeed, such a prima facie case can only be met when **all** of the following requirements are met: (1) the combined references must teach or suggest all the claim limitations; (2) there must be some suggestion or motivation in the references themselves (or in the knowledge available to those skilled in the art) to combine the

references; and (3) there must be a reasonable expectation of success. See MPEP §§ 706.02(j) and 2143. In this case, neither Burridge nor Haefel teaches or discloses a computer process for generating, *before execution of an application requested by a client*, a customized library for execution of the requested application by the client, the computer process comprising, inter alia: 1) comparing the one or more application-referenced types to the one or more client-loaded types to identify one or more client-needed types, wherein client-needed types comprise one or more application-referenced types that are not client-loaded types; or 2) separately sending the customized library and the application to the client for execution.

The present invention includes a customized library management method and system for generating a customized library needed for execution of an application in a client system. In response to an identification of a given application, such as a request from the client system or an internal instruction of the server, the server determines the appropriate types to include in a library to be sent to the client based on certain parameters. The parameters may include, for example, the types referenced by the application; the types already "loaded" on the client system, and a device profile describing characteristics of the client system. The customized library includes a subset of the types, referred to in the exemplary embodiments as the client-needed types, that are required by the application; the client-needed types include all application-referenced types that are not already "loaded" on the client. A type is loaded on a client if the type is present, resides, or is stored at the client. As such, the client-needed types, in the customized library, do not include all application-referenced types, but only those types not already on the client. The customized library is generated and sent to the client before the application is executed. See Application, page 6, lines 10-11 ("The requested application and the customized library are then transmitted to the client for execution.")

Examiner relies heavily on Burridge. Burridge describes methods for optimizing a program during its runtime execution. See col. 2, 45-51 ("a method and apparatus for determining which program units are loaded *during execution* of a dynamically loaded program"). More particularly, Burridge creates a library file of program files "loaded" during execution of a main program unit. See col. 2, lines 56-64 ("creating at least one library file containing only application program files loaded *during the first execution*"). Burridge describes loading as loading a program unit at run time before the unit is used, similar to invoking an

object before executing a contained method in the object. See col. 2, lines 38-39 ("Each reference program unit must be loaded at run time before the referenced unit is used").

To create the library, Burridge describes an offline loader that executes a main program unit. See col. 5, lines 16-18. The offline loader waits for the main program to request a program unit or file, similar to an object requesting another object, and loads the program unit from the client source. See col. 5, lines 19-22 ("If the offline loader 26 requires a program unit that has not already been loaded, the offline loader 26 obtains the program unit from the pathnames specified.") The program unit already exists on the client or can be accessed by the main program already on the client. The offline loader then writes a copy of the program unit to a library. See col. 5, lines 22-25 ("the offline loader 26 writes a copy of the program file to an application library file"). The library file is then used for any subsequent executions of the main program. See col. 5, line 66 – col. 6, line 1 ("The library file created at reference numeral 38 is used as the input source for application program files in subsequent executions of the main program unit."). At a later execution of the main program, the runtime loader obtains the program units from the library file, which reduces the overhead in obtaining application programs from multiple sources. See col. 6, lines 5-9.

Haefel describes Java archives (JAR files). More particularly, Haefel describes deployment descriptors that can be used with JAR files to customize the behavior of software at runtime. See page 30, paragraphs 4-6.

Independent Claim 1, Claim 24, Claim 26, Claim 50, and Claim 51

Amended claims 1, 24, and 26 are allowable over Burridge and Haefel either in combination or alone. Neither Burridge nor Haefel teaches or discloses a computer process for generating, before execution of an application requested by a client, a customized library for execution of the requested application by the client, the computer process comprising, inter alia: 1) comparing the one or more application-referenced types to the one or more client-loaded types to identify one or more client-needed types, wherein client-needed types comprise one or more application-referenced types that are not client-loaded types; or 2) separately sending the customized library and the application to the client for execution. Burridge, the primary prior art reference, and the present invention, as defined in the claims, are fundamentally different. The

differences, and why the present application is allowable over Burridge, will become abundantly clear after the following discussion.

First, Burridge does not teach a computer process for generating, before execution of an application requested by a client, a customized library for execution of the requested application by the client. The present invention, as defined in the claims, comprises generating a customized library *before* execution of a requested application. In one embodiment, the requested application is analyzed by extracting references to types, which are referenced using fully qualified names. See Application, page 8, lines 14-19. Alternatively, "the application-referenced types may be identified by parsing the application binary representation or associated portion thereof." Id. at lines 21-23. In either case, the analysis is done without executing the application. All directly or indirectly referenced types are resolved by applying the method recursively to application-referenced types. See id., lines 19-21.

Burridge does not generate a customized library before execution of a requested application. Instead, Burridge executes the application a first time in what is referred to as a "pre-run" to determine what types to include in the library. See col. 5, lines 17-34 ("[A]n offline loader executes the main method and creates a library file containing all application files loading during execution of the main method.") Further, Burridge may not resolve all possible references during the first run, requiring the run-time loader to "optionally [search] an alternate source" for any application-referenced type not found in the customized library during subsequent executions of the application. See col. 6, lines 18-28.

Next, Burridge does not teach comparing the one or more application-referenced types to the one or more client-loaded types to identify one or more client-needed types, wherein client-needed types comprise one or more application-referenced types that are not client-loaded types. The library in Burridge attempts to include every file needed by the main method. See col. 5, lines 26-28 ("contains all application program files 22 loaded during the execution of the main method."). The library in Burridge essentially contains all application-referenced types. No comparison is made between application-referenced types and those types already loaded on the client. Thus, Burridge does not identify client-needed types as defined in the present invention. While Burridge discloses a run-time loader identifying types that are not in the customized

library at run-time, Burridge does not teach or suggest that application-referenced types are compared to client-loaded types before execution of the application to determine what types to include in the customized library.

The present invention is fundamentally different from Burridge in that a comparison is made, before generating the customized library and before executing the application, between the application-referenced types and the client-loaded types, where client-loaded types are those types already loaded on the client. See Application, page 9, lines 9-11 ("The filter module 214 compares the application-referenced type list 222 against a catalog 216, which in one embodiment contains a description of the types already loaded on the client 202.") Only client-needed types, or those application-referenced types which are not client-loaded types, will be included in the customized library. See id., lines 11-19.

Finally, Burridge does not teach separately sending the customized library and the application to the client for execution. The application, referred to as the main program unit or main method in Burridge, is already present and executing on the client in Burridge. Burridge describes executing the main method that is to be optimized. See col. 5, lines 13-20. ("A user or developer determines which main method will be optimized...loads the main method and begins to execute it."). The main method must be present to execute the main method for optimization. Thus, the main method or application is never sent to a client that requested the application. Also, as described above, Burridge executes the application before generating the customized library. Thus, the application is not sent to a client for execution.

Further, the library in Burridge is also not sent to a client that requested the application. The library file is a collocation of files used by an application. See col. 11, lines 42-43 ("a novel method and apparatus for collocating dynamically loaded program files has been described."). The library file provides the system to decrease the memory footprint of the running application. See generally col. 2, lines 48-51. As such, the library file of Burridge is stored locally in the memory of the client executing the main method. The library file is never sent to a client. Again, Burridge executes the application once before generating the customized library; thus, the library is not sent to a client prior to execution of the application.

In contrast, the present invention, as defined in the claims, sends both the application and the customized library to a client that requested the application so the client can execute the application. Burridge does not describe any communication between two computer systems involving the application and the library, nor does Burridge describe providing the customized library and application to a client for execution by the client. Therefore, Burridge simple does not perform the same process as the present invention.

The present invention, as defined in the claims, and Burridge are not two interchangeable methods that do the same thing, but two separate methods that execute separate functions for separate purposes. For example, a user may request an application using the present invention, as defined in the claims. Before the application is executed, the computer process can determine what types are on the client and compare those types to the application-referenced types. The application and a customized library, with application-referenced types not on the client, can then be sent to the client requester for execution. Once the application is stored at the client, a user could use the process described in Burridge to optimize the application. For example, the application could be executed. As types are loaded, either from the client-loaded types or the client-needed types in the customized library, the offline loader can place the types in another library. The application could then use the other library thereafter. As can be understood by this example, Burridge and the present invention, as defined in the claims, simply accomplish different things and are not comparable.

Haefel does not overcome the shortcomings of Burridge. Haefel describes deployment descriptors that can be used with JAR files to customize the behavior of software at runtime. See page 30, paragraphs 4-6. There is no mention in Haefel about a computer process for generating, before execution of an application requested by a client, a customized library for execution of the requested application by the client, the computer process comprising, inter alia: 1) comparing the one or more application-referenced types to the one or more client-loaded types to identify one or more client-needed types, wherein client-needed types comprise one or more application-referenced types that are not client-loaded types; or 2) separately sending the customized library and the application to the client for execution. Therefore, Haefel does not teach these elements of the claims.

While Haefel does describe using JAR files to shrink wrap third party components, there is no motivation to combine Haefel and Burridge. To the contrary, Burridge teaches away from the combination. Burridge desires to collocate program units in a single library. See col. 4, lines 40-42. Haefel describes packaging components for delivery to third parties. See page 90, paragraph 1. The combination of Burridge and Haefel would result in two libraries, a library or JAR file created for optimization and the JAR file received from a third party. The combination of the two libraries is contrary to the goal of Burridge, and thus, Burridge teaches away from this combination.

Further, Burridge does not teach sending the library or JAR file but keeping the library at the client to reduce memory allocation and optimize the function of the main method. See col. 2, lines 48-53. The combination of Burridge and Haefel suggests creating a JAR file using the method described in Burridge and sending the JAR file to another location. By sending the JAR file to another location, the purpose of Burridge is defeated, i.e., trying to optimize the execution of an application at the client by collocating files at the client. For these reasons, there is no motivation to combine the references.

For the reasons stated above, independent claims 1, 24, 26, 50, and 51 are allowable over the prior art. Likewise, claims 2, 4, 6-22, 27, and 29-33 depend from the allowable independent claims and thus are also allowable. Examiner is respectfully requested to remove the rejections to the amended claims and allow all claims now present.

Conclusion

This Amendment fully responds to the Final Office Action mailed on February 2, 2006. It is recognized that the Office Action may contain arguments and rejections that are not directly addressed by this Amendment due to the fact that they are rendered moot in light of the preceding arguments and amendments in favor of patentability. Hence, the failure, if any, of this Amendment to directly address an argument raised by the Examiner should not be interpreted as reflecting the Applicant's belief that such argument has merit. Furthermore, the claims of the present application may include other elements, not discussed in this Amendment, which are not shown, taught, or otherwise suggested by the art of record. Accordingly, the preceding arguments in favor of patentability are advanced without prejudice to other bases of patentability.

It is believed that no further fees are due with this Response. However, the Commissioner is hereby authorized to charge any deficiencies or credit any overpayment with respect to this patent application to deposit account number 13-2725.

In light of the above amendments and remarks, it is believed that the application is now in condition for allowance, and such action is respectfully requested. If the Examiner believes a telephone conference would advance the prosecution of this application, the Examiner is invited to telephone the undersigned at the below-listed telephone number.

Respectfully submitted,

Date: August 21, 2006

Tadd F. Wilson
Reg. No. 54,544
MERCHANT & GOULD P.C.
P.O. Box 2903
Minneapolis, Minnesota 55402-0903
303.357.1651